پروژه ششم علوم اعصاب محاسباتی

- با توجه به تغییرات عمده اعمال شده در طراحی و رابط کاربری فریم‌ورک، در این تمرین کدها حذف نشده‌اند تا لطف کنید و در بازخورد، درباره این رابط و ساختار نیز بازخورد دهید. با تشکر.

0. فهرست مطالب

1. ساخت شبکه
2. بررسی طرح آزمایش‌ها
3. روش STDP
4. تأثیر پارامتر ثابت زمانی در روش STDP
5. تأثیر پارامتر اندازه گام آموزشی در STDP
6. تآثیر مقداردهی اولیه وزن‌ها
7. تآثیر اضافه شدن کاهش وزنی
8. روش Flat-STDP
9. تأثیر پارامتر پنجره Flat-STDP

In [1]:
```
import warnings
warnings.filterwarnings("ignore")
import torch
```

1. ساخت شبکه

با توجه به هدف تمرین، در این تمرین فقط تغییرات مربوط به قوانین یادگیری و ویژگی خاص مقدار دهی اولیه وزن‌ها (که با توجه به بدون ناظر بودن یادگیری، می‌تواند بسیار موثر باشد) را بررسی می‌کنیم.
بنابراین، می‌توانیم باقی شبکه را بسازیم و با استفاده از یک تابع کمکی، این دو عنصر را هنگام آزمایش تعیین کنیم.

In [2]:
```
from cnsproject.network.network import Network
from cnsproject.network.network_wrapper import FROM,TO,OF,FOLLOWING
from cnsproject.network.neural_populations import LIFPopulation
from cnsproject.network.encoders import PoissonEncoder
from cnsproject.network.axon_sets import SimpleAxonSet
from cnsproject.network.dendrite_sets import SimpleDendriteSet
from cnsproject.network.synapse_sets import SimpleSynapseSet
from cnsproject.network.connectivity_patterns import dense_connectivity

def build_net(w_init, learning_rule_enforcer):
    net = Network(dt=1.)
    net += PoissonEncoder('encoder', shape=im1.shape, max_input=255)
    net += LIFPopulation('output', (2,))
    net += (
        SimpleSynapseSet(name='synapse', connectivity=dense_connectivity())
    ) |FROM| (
        SimpleAxonSet(scale=5)
        |OF| net['encoder']
    ) |TO| (
        SimpleDendriteSet(w=w_init)
        |OF| net['output']
    ) |FOLLOWING| (
```

```
        learning_rule_enforcer
    )
    net.reset()
    return net
```

2. بررسی طرح آزمایش‌ها

از دو تصویر ساده زیر استفاده خواهیم کرد:

In [3]:
```
import matplotlib.pyplot as plt
from cnsproject.monitors.plotter import Plotter
from PIL import Image
import numpy as np

path1 = "+.jpg"
path2 = "o.jpg"

plt.figure(figsize=(14,2))
p = Plotter([
    ['im1','nim1',None,'im2','nim2'],
])
p.imshow('im1', Image.open(path1).convert('L'), title="Original")
p.imshow('im2', Image.open(path2).convert('L'), title="Original")

def normalize(path):
    im = np.array(Image.open(path).convert('L'))
    im = torch.from_numpy(im)
    im = 255 - im
    im = im.float()
    im /= im.float().sum()
    im -= im.min()
    im /= im.max()
    im *= 255
    return im

im1,im2 = normalize(path1),normalize(path2)
p.imshow('nim1', Image.fromarray(im1.byte().numpy()), title="Normalized")
p.imshow('nim2', Image.fromarray(im2.byte().numpy()), title="Normalized")
plt.show()
```

دو تصویر بالا را به صورت متناوب به شبکه ورودی خواهیم داد. بین هر تعویض مدتی محدود فاصله زمانی با مقدار کمی نویز قرار خواهیم داد:

In [19]:
```
from cnsproject.network.weight_initializations import norm_initialization
from cnsproject.learning.learning_rule_enforcers import NoOp
from cnsproject.monitors.monitors import Monitor

def simulate(net, step_time = 150, step_count = 8):
    monitor = Monitor(net, state_calls={
        'encoder_spikes': net['encoder'].spikes,
        'output_spikes': net['output'].spikes,
    }, time=step_time)
    dendrite_monitor = Monitor(net[[['synapse']]].dendrite, state_variables=[
    monitor.reset(), dendrite_monitor.reset()
```

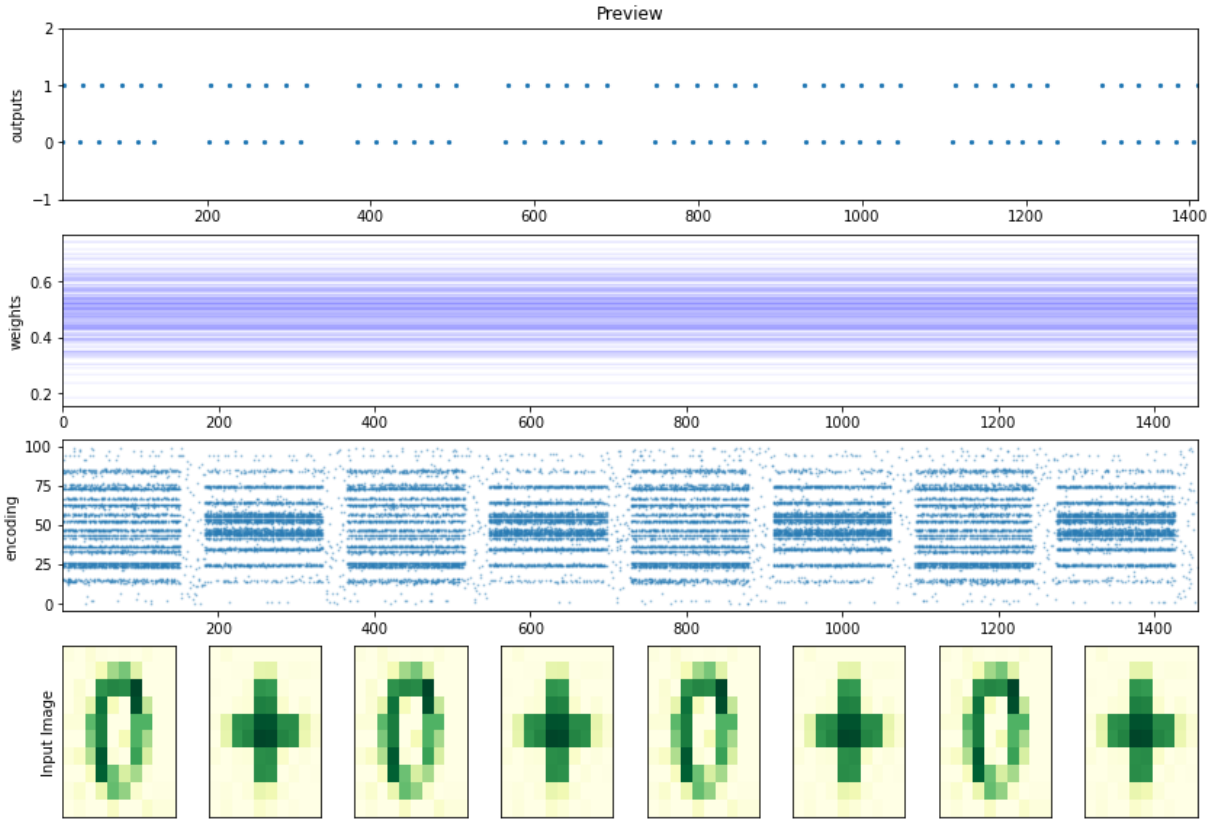
```
    net.reset()
    ims = [im1,im2]
    for i in range(step_count):
        net['encoder'].encode(ims[i%2==0])
        monitor.simulate(net.run, attendance=[dendrite_monitor], reset=False)
        net['encoder'].encode(im1*0)
        monitor.simulate(net.run,
                         inputs={"encoder_clamp": torch.rand(int(step_time//5
                         attendance=[dendrite_monitor],
                         time=int(step_time//5), reset=False)
    return monitor,dendrite_monitor


net = build_net(norm_initialization(), NoOp())
monitor,dendrite_monitor = simulate(net)
```

In [20]:

```
def plot_every_thing(monitor, dendrite_monitor, steps = 8, name = ''):
    ims = [im1,im2]
    plt.figure(figsize=(14,10))
    p = Plotter([
        ['output']*steps,
        ['weights']*steps,
        ['encode']*steps,
        [f'im{i}' for i in range(steps)],
    ], monitor=monitor, wspace=0.3)
    for i in range(steps):
        y_label = "Input Image" if i==0 else ""
        p.imshow(f'im{i}', ims[i%2==0], y_label=y_label, cmap='YlGn', interpo
    p.population_activity_raster('encode', y='encoder_spikes', y_label='encoc
    p.population_plot('weights', y='w', y_label="weights", monitor=dendrite_m
                      population_alpha=.05, x_lim='fit')
    p.population_activity_raster('output', y='output_spikes', y_label='output
                                 x_lim='fit', title=name)
    p.show()

plot_every_thing(monitor,dendrite_monitor,name='Preview')
```



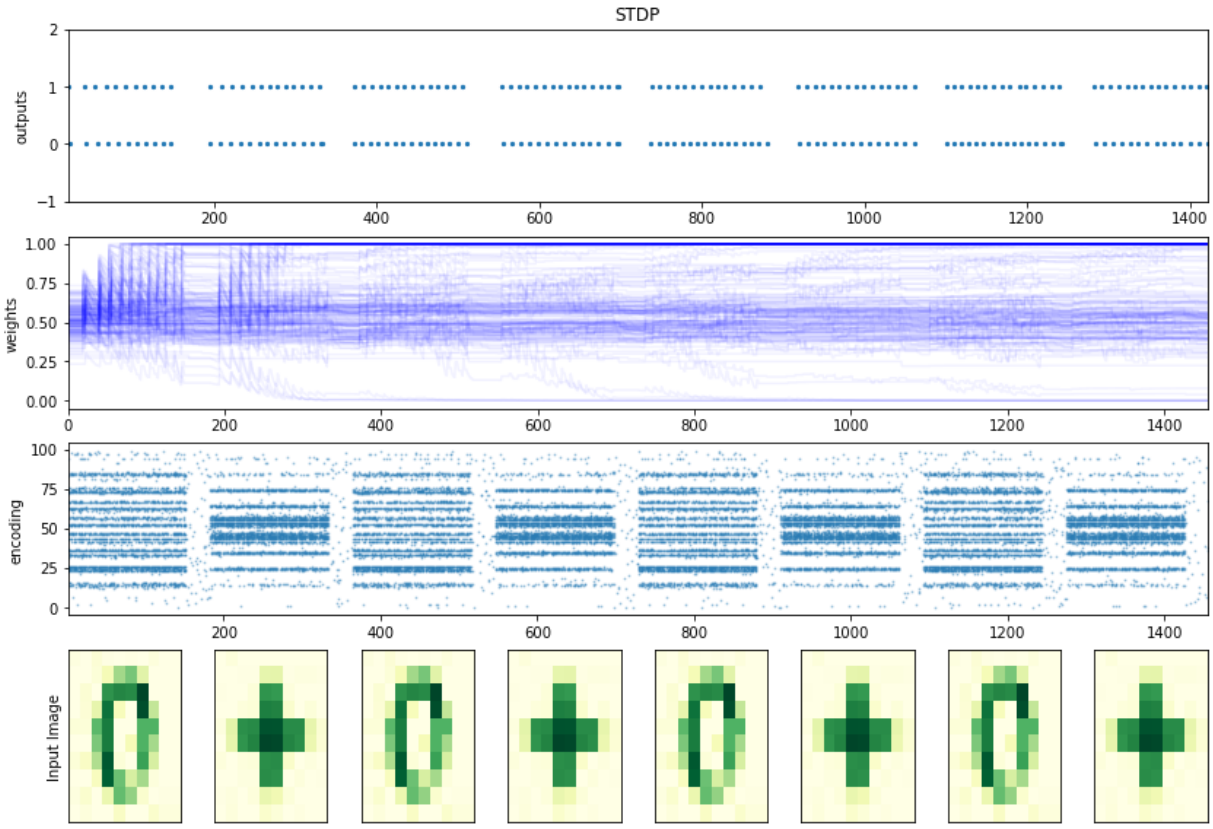
همانطور که مشاهده می‌کنیم، بین هر دو تصویر، مقداری اسپایک رندوم وجود دارد. همچنین الگوی

اسپایک‌های ورودی به هنگام هر تصویر کاملا متفاوت است. همچنین شاهده وضعیت ثابت وزن‌ها و خروجی مشابه از طرف هر دو نورون هستیم. در ادامه آزمایش تغییر وزن‌ها و تغییر الگوهای خروجی را بررسی خواهیم کرد.

3. روش STDP

از مشاهده عملکرد این روش شروع می‌کنیم.

In [21]:
```
from cnsproject.learning.learning_rule_enforcers import STDP
from cnsproject.learning.synaptic_taggers import STDPST
from cnsproject.learning.learning_rates import stdp_wdlr
net = build_net(norm_initialization(),
                STDP(ltp_wdlr=stdp_wdlr(.1), ltd_wdlr=stdp_wdlr(.1))
               )
monitor,dendrite_monitor = simulate(net)
plot_every_thing(monitor,dendrite_monitor,name='STDP')
```

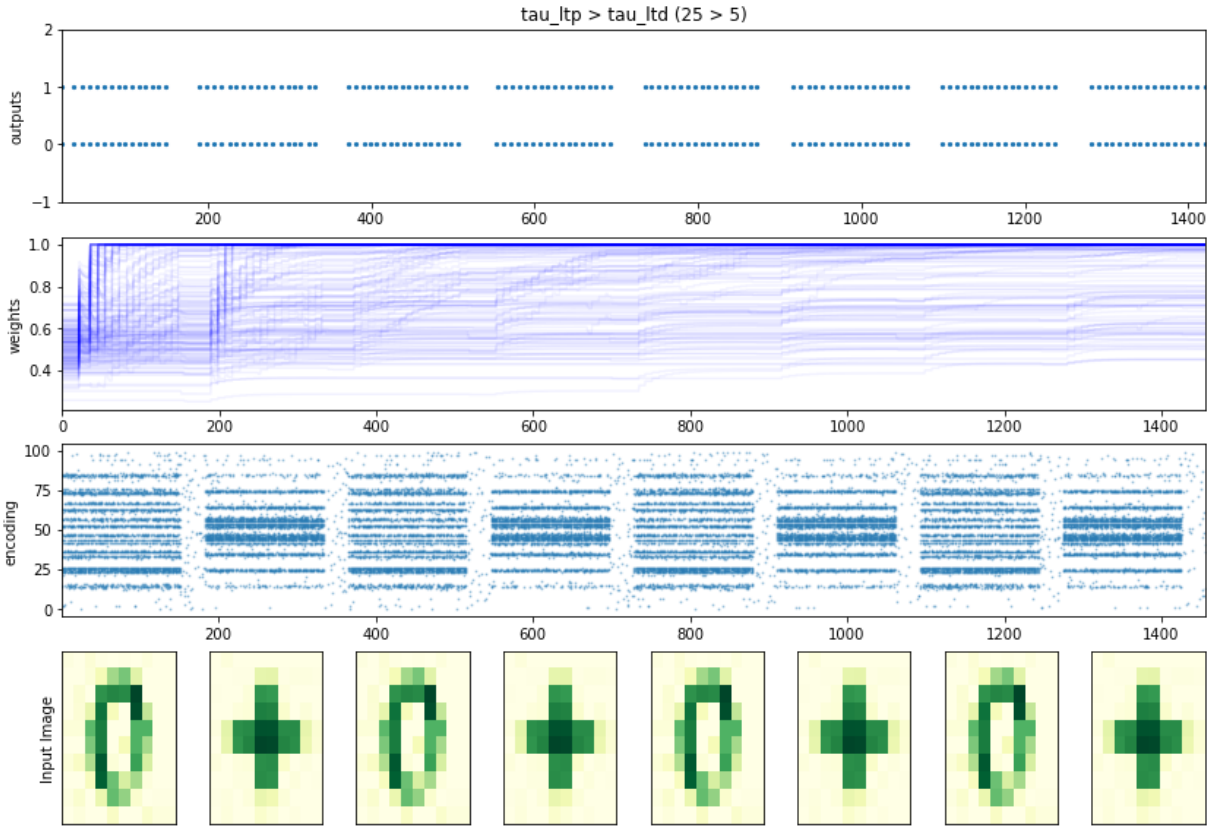


مشاهده می‌کنیم وزن‌ها در جهت‌های مثبت و منفی حرکت می‌کنند. نکات زیر قابل مشاهده است:
- وزن‌ها از صفر پایین‌تر و از یک فراتر نمی‌روند چون محدودیت وزن در دندریت‌ها وجود دارد.
- تغییرات وزنی در نواحی نزدیک به صفر و یک کندتر است چرا که از گام‌های آموزشی وابسته به وزن استفاده شده است. این مسئله در بخش گام‌های آموزشی بیشتر بررسی خواهد شد.
- تعدادی از وزن‌ها روند صعود یا نزول خود را بدون توجه به تصویر ورودی حفظ می‌کنند. این وزن‌ها احتمالا مربوط به پیکسل‌های مشترک در دو تصویراند.
- مشاهده می‌شود که برخی وزن‌ها فقط در صورت مشاهده برخی عکس‌ها تغییر می‌کنند. مشخصا این وزن‌ها مربوط به پیکسل‌هایی هستند که آن تصویر نسبت به تصویر دیگر در آن پیکسل‌ها روشن‌تر (یا اگر وزن درحال کاهش است، تاریک‌تر) است.
- تغییرات هنگام نویز قابل چشم‌پوشی است.
- اسپایک‌های خروجی در بازه‌های مشاهده عکس تجمیع شده اما هنوز تفاوت چشم‌گیری در الگوی رفتاری دو نورون خروجی در مواجهه با عکس های ورودی متفاوت دیده نمی‌شود.

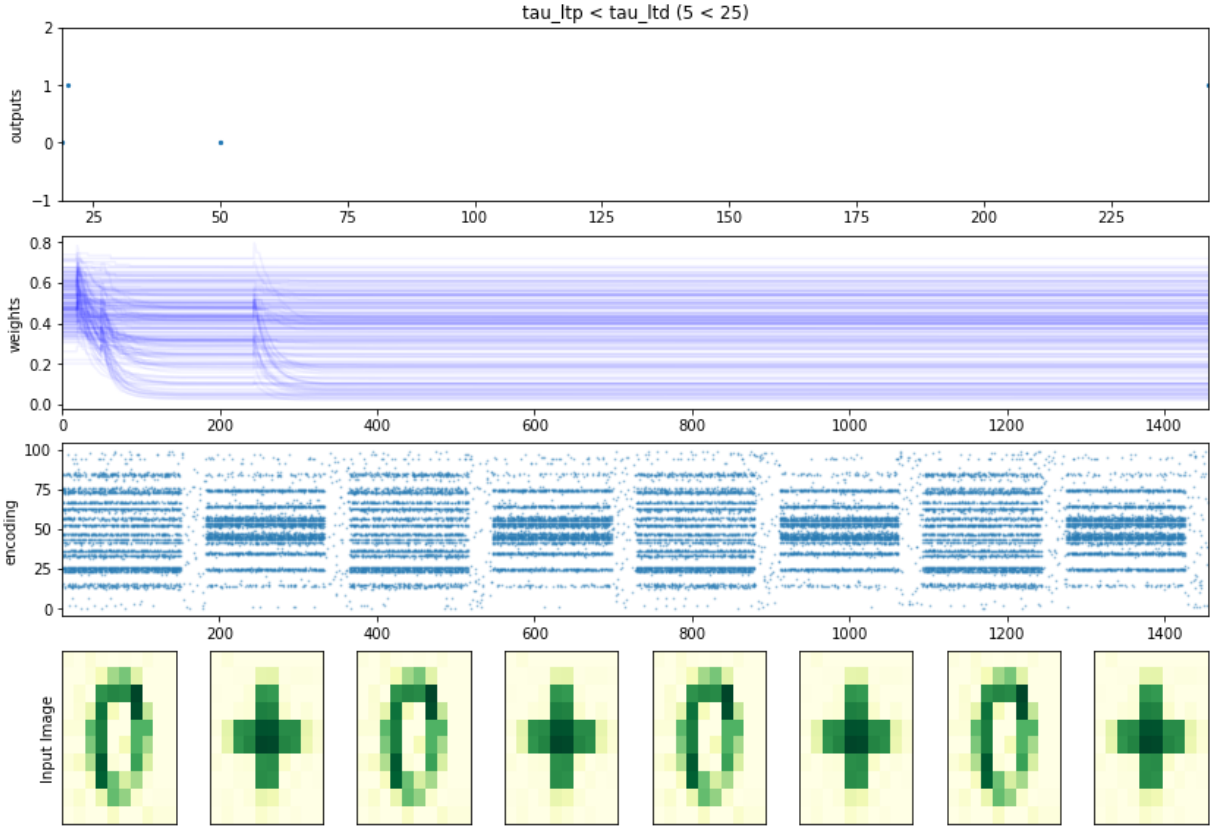
4. تاثیر پارامتر ثابت زمانی در روش STDP

انتظار داریم با زیاد کردن ثابت زمانی مربوط به هر بخش (ltp یا ltd / افزایش یا کاهش / pre یا post)، تغییرات وزنی مربوط به آن بخش شدیدتر صورت بگیرد چرا که با این کار، پایداری اثرات اسپایکی زیاد شده و زمان به روز رسانی وزن‌ها، اثر بیشتری را حاصل می‌کنند.

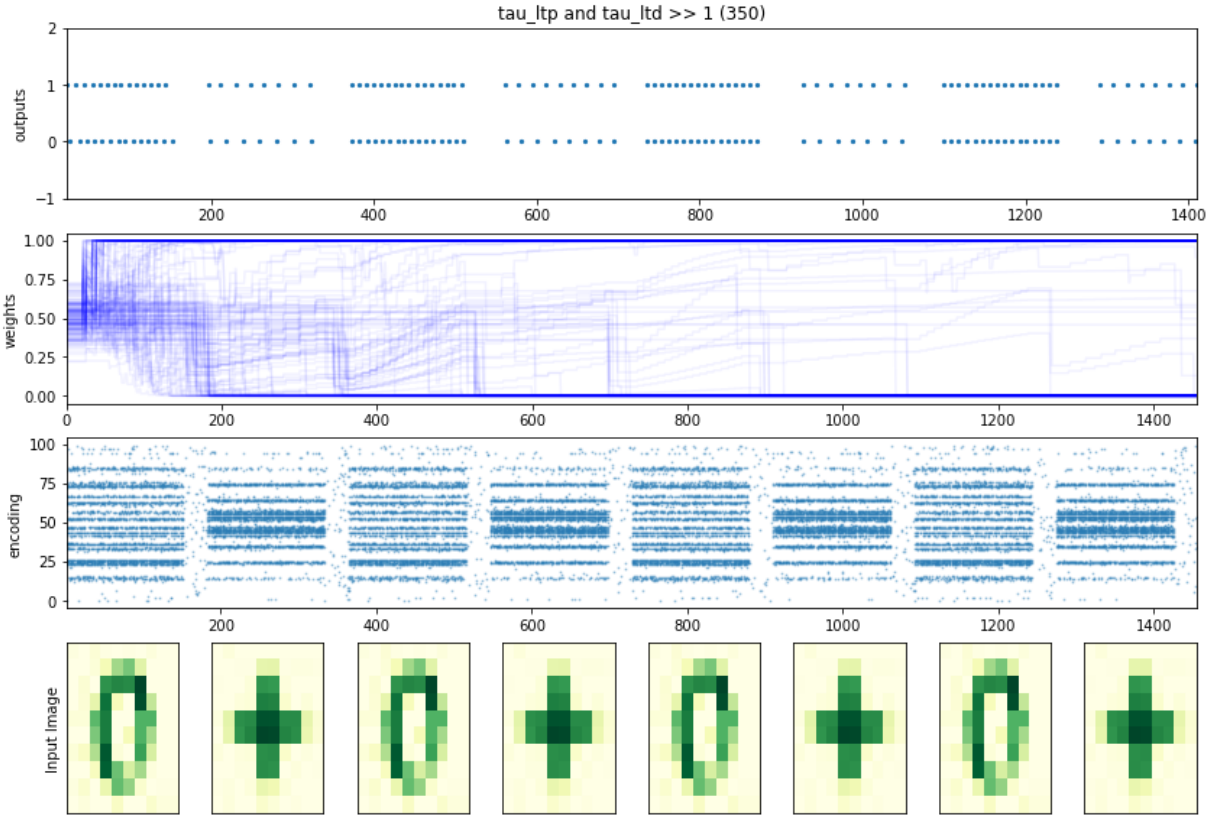
In [22]:
```
net = build_net(norm_initialization(),
    STDP(
        pre_traces=STDPST(tau=25., scale=1.),
        post_traces=STDPST(tau=5., scale=1.),
        ltp_wdlr=stdp_wdlr(.1),
        ltd_wdlr=stdp_wdlr(.1)
    )
)
monitor,dendrite_monitor = simulate(net)
plot_every_thing(monitor,dendrite_monitor, name="tau_ltp > tau_ltd (25 > 5)")
```



In [23]:
```
net = build_net(norm_initialization(),
    STDP(
        pre_traces=STDPST(tau=5., scale=1.),
        post_traces=STDPST(tau=25., scale=1.),
        ltp_wdlr=stdp_wdlr(.1),
        ltd_wdlr=stdp_wdlr(.1)
    )
)
monitor,dendrite_monitor = simulate(net)
plot_every_thing(monitor,dendrite_monitor, name="tau_ltp < tau_ltd (5 < 25)")
```



In [26]:
```
net = build_net(norm_initialization(),
    STDP(
        pre_traces=STDPST(tau=350., scale=1.),
        post_traces=STDPST(tau=350., scale=1.),
        ltp_wdlr=stdp_wdlr(.1),
        ltd_wdlr=stdp_wdlr(.1)
    )
)
monitor,dendrite_monitor = simulate(net)
plot_every_thing(monitor,dendrite_monitor, name="tau_ltp and tau_ltd >> 1 (35
```



نتایج مورد انتظار گفته شده را در دو نمودار اول شاهد هستیم. هر بخش که دارای ثابت زمانی بزرگ‌تری باشد، آموزش شبکه به آن سمت منعطف می‌شود. اگر ltd قوی‌تر باشد، خروجی میرا می‌شود.
در نمودار سوم نتایج جالبی را شاهد هستیم. می‌بینیم که تفاوت‌هایی در الگوی خروجی به ازای ورودی متفاوت درحال ظاهر شدن است (هرچند این الگو بین دو نورون مشترک است). دلیل این امر تسریع فرایند آموزش با tau بزرگ‌تر است. همچنین جالب است که توجه کنید نظم تغییرات وزن‌ها کم شده. دلیل این امر آن است که tau بسیار بزرگ باعث شده میزان اطلاعات غلط آموزش دیده زیاد شود و مدل پس از مدتی متوجه این اشتباه شده و سعی در اصلاح آن‌ها کند.

5. تأثیر پارامتر اندازه گام آموزشی در STDP

در این بخش، از tau=100 استفاده خواهیم کرد. فقط از گام‌های آموزشی وابسته به زمان استفاده خواهیم کرد که از رابطه‌ی زیر پیروی می‌کنند:

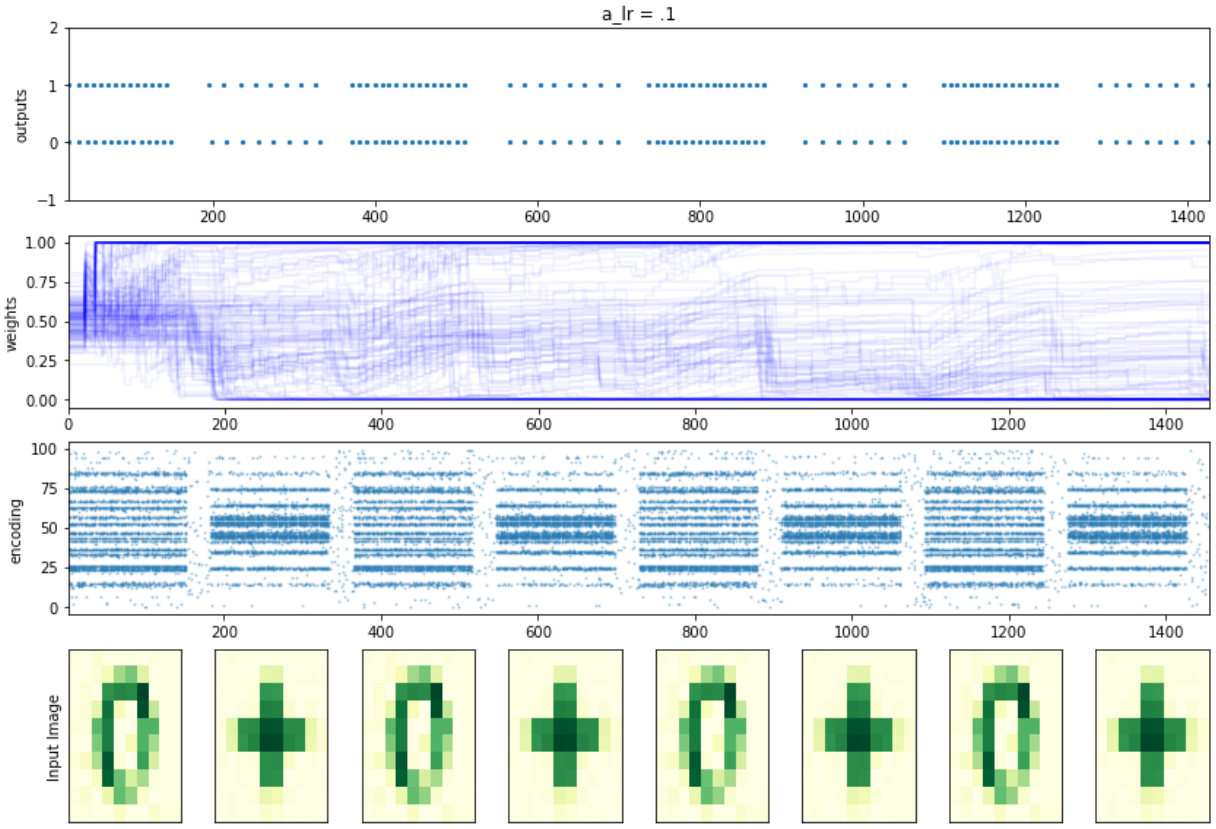
$$lr = a_{lr} \times (w_{max} - w) \times (w - w_{min})$$

عملکرد شبکه را با مقادیر مختلف برای a بررسی خواهیم کرد.

In [27]:
```
net = build_net(norm_initialization(),
    STDP(
        pre_traces=STDPST(tau=100., scale=1.),
        post_traces=STDPST(tau=100., scale=1.),
        ltp_wdlr=stdp_wdlr(1),
        ltd_wdlr=stdp_wdlr(1)
    )
)
monitor,dendrite_monitor = simulate(net)
plot_every_thing(monitor,dendrite_monitor, name="a_lr = 1")
```

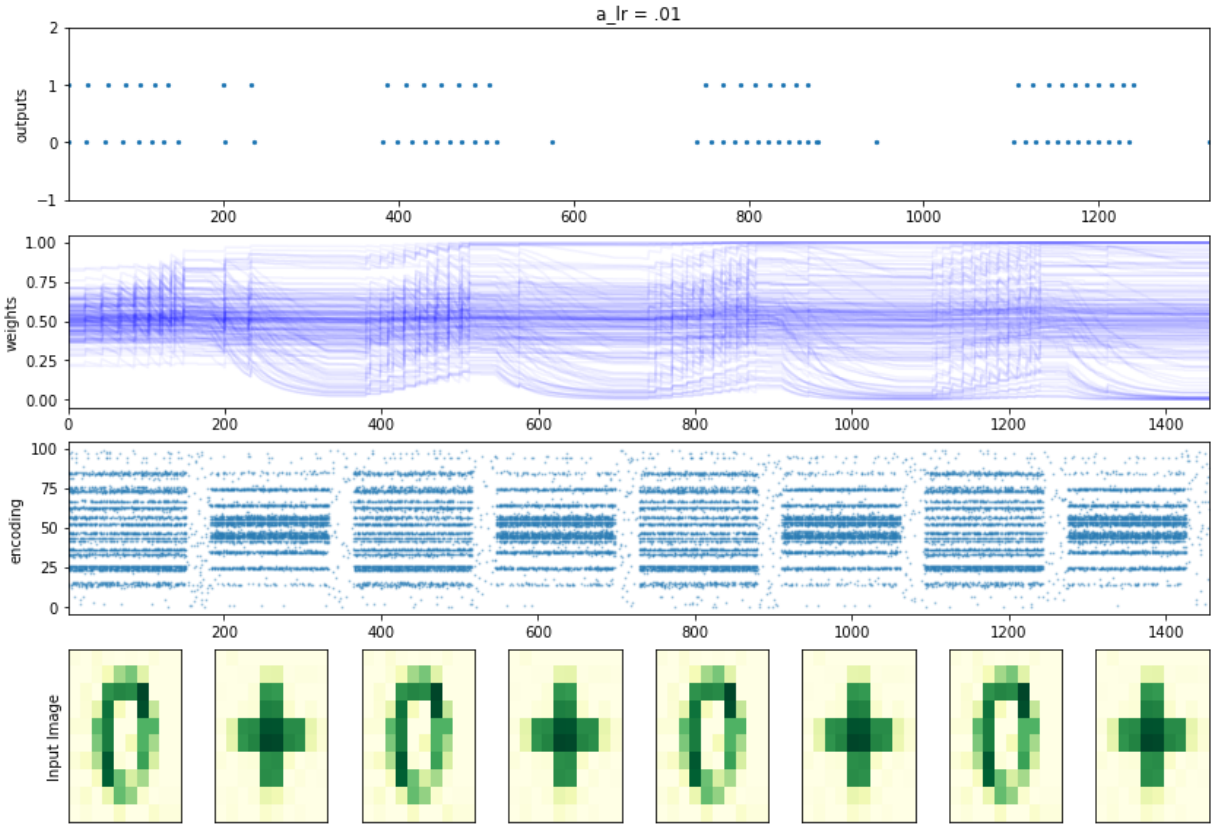
In [28]:
```
net = build_net(norm_initialization(),
    STDP(
```

```
        pre_traces=STDPST(tau=100., scale=1.),
        post_traces=STDPST(tau=100., scale=1.),
        ltp_wdlr=stdp_wdlr(.1),
        ltd_wdlr=stdp_wdlr(.1)
    )
)
monitor,dendrite_monitor = simulate(net)
plot_every_thing(monitor,dendrite_monitor, name="a_lr = .1")
```



In [31]:

```
net = build_net(norm_initialization(),
    STDP(
        pre_traces=STDPST(tau=100., scale=1.),
        post_traces=STDPST(tau=100., scale=1.),
        ltp_wdlr=stdp_wdlr(.01),
        ltd_wdlr=stdp_wdlr(.01)
    )
)
monitor,dendrite_monitor = simulate(net)
plot_every_thing(monitor,dendrite_monitor, name="a_lr = .01")
```

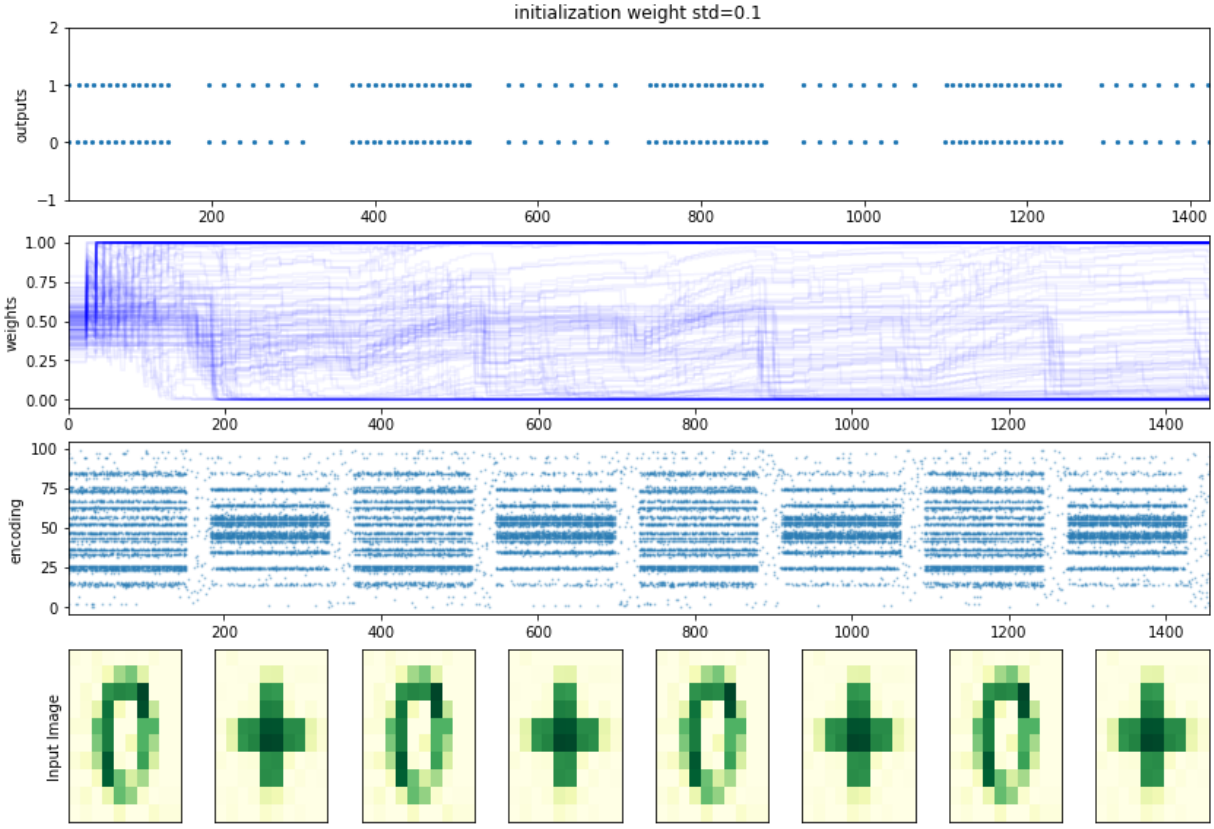


مشاهده می‌کنیم با گام آموزشی بزرگ، تغییرات شدیدتر و نا دقیق‌تر می‌شوند. در گام‌های کوچک، تفاوت‌های عکس‌ها بهتر درک شده است. این اثر مانند تمامی زمینه‌های دیگر یادگیری ماشین می‌باشد.

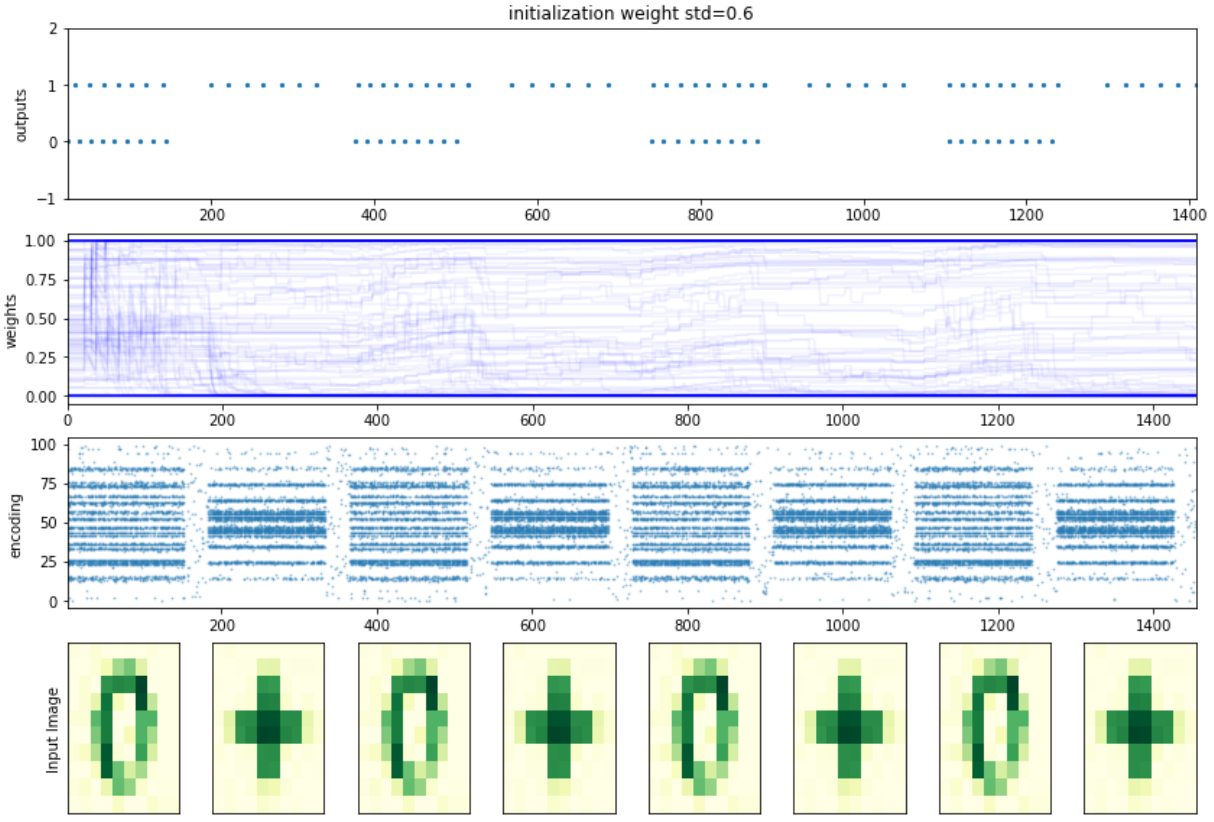
6. تآثیر مقداردهی اولیه وزن‌ها

می‌دانیم که تفاوت در الگوی خروجی به ازای ورودی‌های متفاوت، حاصل تشدید تفاوت‌های اولیه موجود در اتصالات است. بنابراین انتظار می‌رود اگر میزان رندوم‌نس این مقدار دهی اولیه بیشتر باشد، شبکه پتانسیل بیشتری برای ارائه الگوهای متفاوت‌تر در خروجی داشته باشد. از tau=100 و a_{lr}=0.1 استفاده خواهیم کرد.

In [37]:
```
net = build_net(norm_initialization(w_std=0.1),
    STDP(
        pre_traces=STDPST(tau=100., scale=1.),
        post_traces=STDPST(tau=100., scale=1.),
        ltp_wdlr=stdp_wdlr(.1),
        ltd_wdlr=stdp_wdlr(.1)
    )
)
monitor,dendrite_monitor = simulate(net)
plot_every_thing(monitor,dendrite_monitor, name="initialization weight std=0.
```



In [42]:
```
net = build_net(norm_initialization(w_std=0.6),
    STDP(
        pre_traces=STDPST(tau=100., scale=1.),
        post_traces=STDPST(tau=100., scale=1.),
        ltp_wdlr=stdp_wdlr(.1),
        ltd_wdlr=stdp_wdlr(.1)
    )
)
monitor,dendrite_monitor = simulate(net)
plot_every_thing(monitor,dendrite_monitor, name="initialization weight std=0.
```

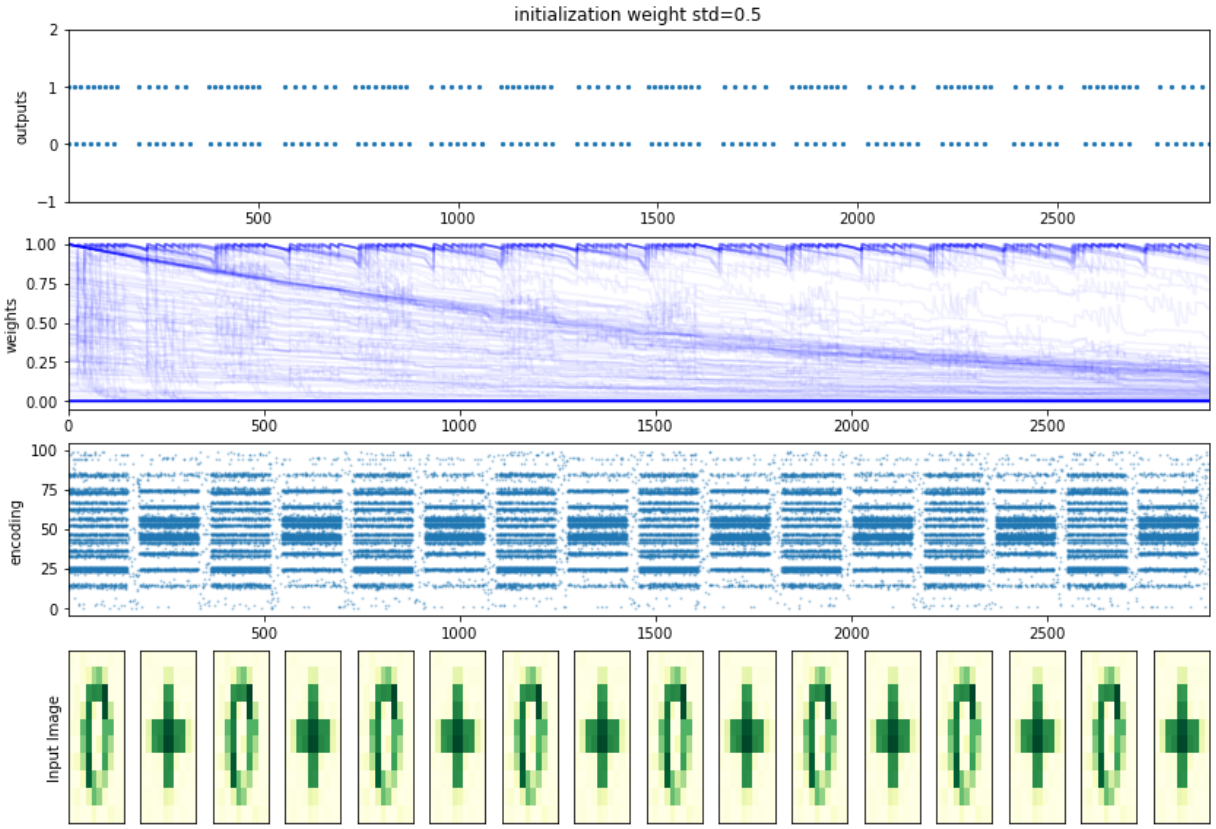


مشاهده می‌کنیم که حدس ما کاملا درست بود (حدس در ابدای بخش آورده شد).

7. تأثیر اضافه شدن کاهش وزنی

در این بخش، کاهش ممتد وزن‌ها را به فرایند یادگیری اضافه می‌کنیم. این تغییر را بر شبکه قبلی (آخرین شبکه بخش قبل) اما با tau=25 اعمال خواهیم کرد. این کاهش وزن با شدت 0.0005 اعمال خواهد شد. چون هدف از این اثر، حذف وزن‌های بی هدف است، این مقدار نیز کفایت می‌کند. شبیه‌سازی را به مدت 16 دوره (دو برابر قبل) ادامه خواهیم داد تا تأثیر مذکور بهتر مشاهده شود.

In [48]:
```
from cnsproject.learning.learning_rule_enforcers import SimpleWeightDecayLRE
net = build_net(norm_initialization(w_std=0.6),
    SimpleWeightDecayLRE(decay=0.0005) +\
    STDP(
        pre_traces=STDPST(tau=25., scale=1.),
        post_traces=STDPST(tau=25., scale=1.),
        ltp_wdlr=stdp_wdlr(.1),
        ltd_wdlr=stdp_wdlr(.1)
    )
)
monitor,dendrite_monitor = simulate(net, step_count=16)
plot_every_thing(monitor,dendrite_monitor, name="initialization weight std=0.
```

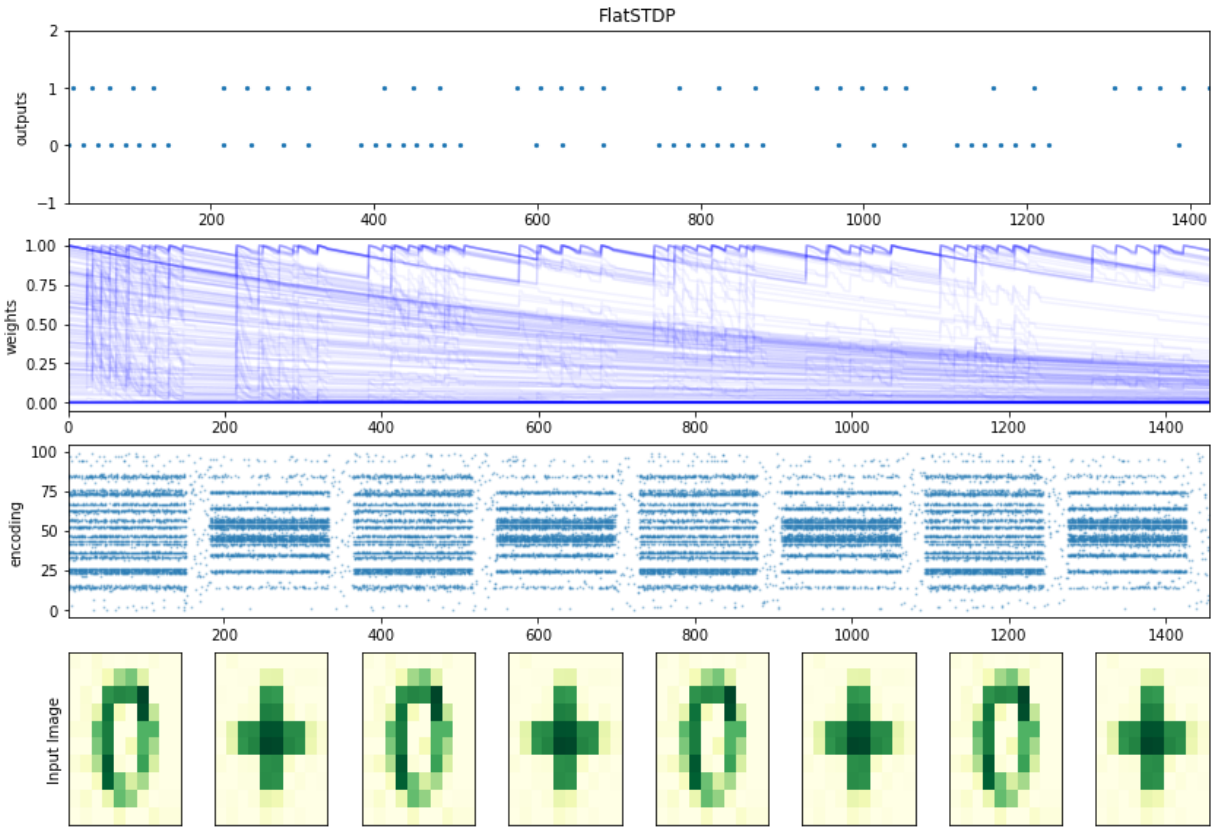


می‌بینیم وزن‌های بی‌اثر که پیش‌از این ثابت بودند به آرامی کاهش پیدا می‌کنند و به نوعی فضا را برای وزن‌های موثر خالی می‌کنند. در این شکل بهتر می‌توان شاهد وزن‌های مختص به هر تصویر بود.

8. روش Flat-STDP

In [50]:
```
from cnsproject.learning.learning_rule_enforcers import FlatSTDP
from cnsproject.learning.synaptic_taggers import FSTDPST
net = build_net(norm_initialization(w_std=.5),
                SimpleWeightDecayLRE(decay=0.001) +\
                FlatSTDP(pre_time=20, post_time=20,
```

```
                            ltp_wdlr=stdp_wdlr(.1),
                            ltd_wdlr=stdp_wdlr(.1))
                )
monitor,dendrite_monitor = simulate(net)
plot_every_thing(monitor,dendrite_monitor,name='FlatSTDP')
```

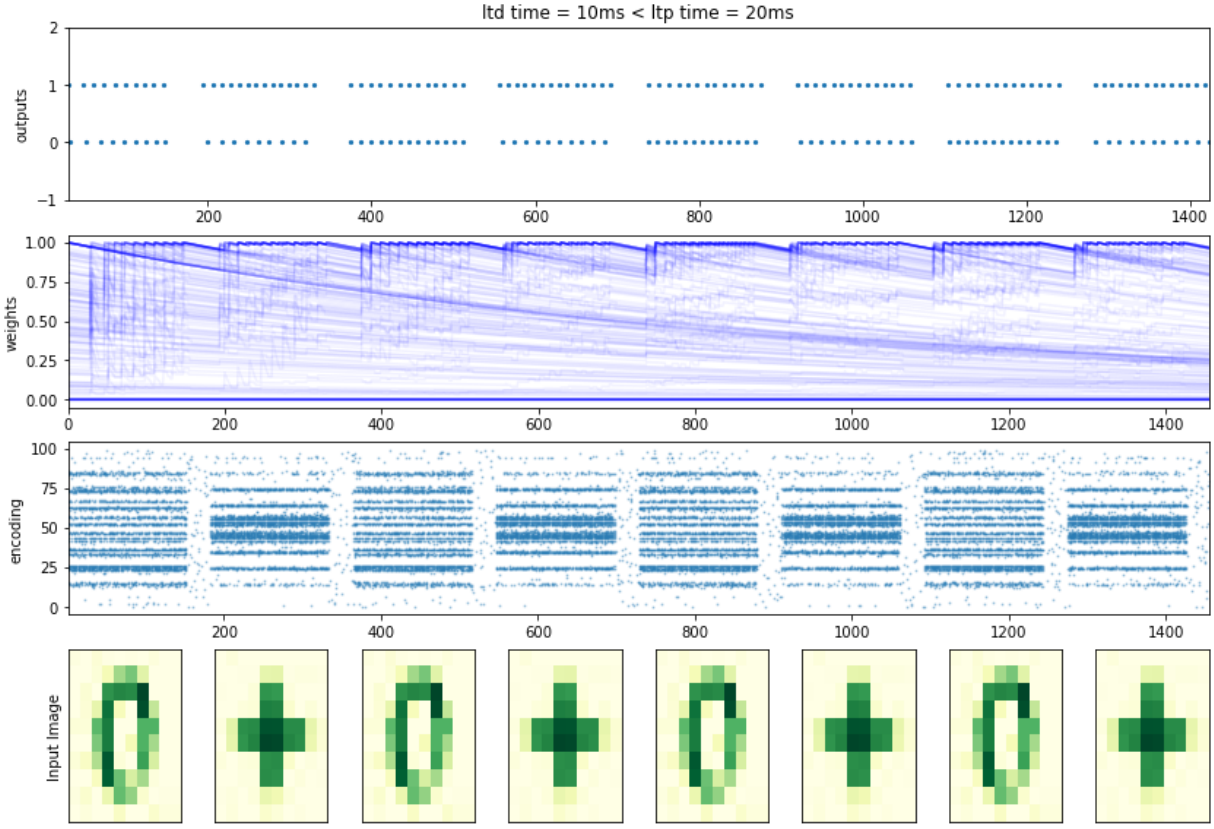


شاهد نتایجی مشابه آنچه قبلا وجود داشت هستیم. دلیل این شباهت سادگی بیش از اندازه آزمایش است. تفاوتی که دیده می‌شود شدت بیشتر تغییرات است که به دلیل انباشته‌ای بودن اثرات اسپایکی در این مدل است.

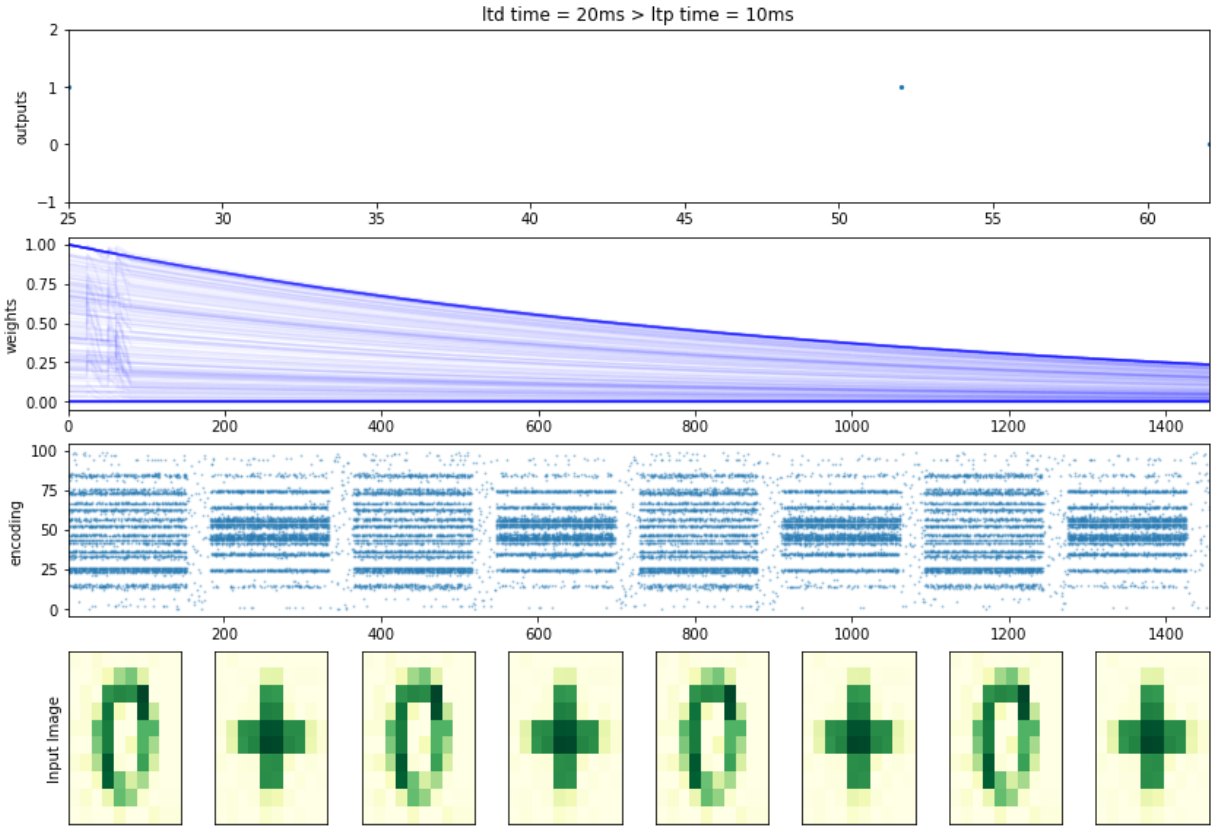
9. تأثیر پارامتر پنجره Flat-STDP

انتظار می‌رود این اثر مانند اثر tau در STDP باشد.

In [52]:
```
net = build_net(norm_initialization(w_std=.5),
                SimpleWeightDecayLRE(decay=0.001) +\
                FlatSTDP(pre_time=20, post_time=10,
                            ltp_wdlr=stdp_wdlr(.1),
                            ltd_wdlr=stdp_wdlr(.1))
                )
monitor,dendrite_monitor = simulate(net)
plot_every_thing(monitor,dendrite_monitor,name='ltd time = 10ms < ltp time =
```



In [53]:
```
net = build_net(norm_initialization(w_std=.5),
                SimpleWeightDecayLRE(decay=0.001) +\
                FlatSTDP(pre_time=10, post_time=20,
                         ltp_wdlr=stdp_wdlr(.1),
                         ltd_wdlr=stdp_wdlr(.1))
               )
monitor,dendrite_monitor = simulate(net)
plot_every_thing(monitor,dendrite_monitor,name='ltd time = 20ms > ltp time =
```



شاهد نتایج مورد انتظار هستیم.